**Федеральное агентство связи**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра Математической кибернетики и информационных технологий

**Отчет по лабораторной работе**

по предмету «СИАОД»

на тему:

«Сетевые алгоритмы. Динамические алгоритмы поиска путей.»

Выполнил: студент группы

Митрохин Ярослав Игоревич

Руководитель:

Кутейников Иван Александрович

Москва 2020

*Цель работы:* реализовать алгоритм поиска кратчайшего расстояния между двумя вершинами ориентированного взвешенного графа. Предусмотреть задание графа в виде матрицы смежности\инциндентности, читаемой из файла. Разработать графический интерфейс пользователя с визуализацией графа и отображением кратчайшего расстояния между задаваемыми пользователем вершины.

По результатам работы проанализировать временную сложность работы заданного алгоритма в зависимости от числа узлов и ребер графа.

*Ход работы:*

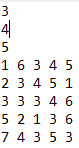
*Код программы:*

import javax.swing.\*;  
import java.awt.\*;  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
import java.sql.SQLOutput;  
import java.util.\*;  
  
import static javafx.scene.input.KeyCode.*X*;  
  
class Levit extends JComponent{  
  
 public void paint (Graphics g)  
 {  
  
 }  
  
 public static int[][] schet(int[][] arr,int vNum) {  
 int INF = Integer.*MAX\_VALUE* / 2;  
 int[][] dist = new int [vNum][vNum];  
 int[] prev = new int [vNum];  
 for (int i=0;i<vNum;i++) System.*arraycopy*(arr[i],0,dist[i],0,vNum);  
 for (int k=0;k<vNum;k++)  
 for (int i2 =0; i2<vNum;i2++)  
 for (int j =0;j<vNum;j++)  
 dist[i2][j]=Integer.*min*(dist[i2][j],dist[i2][k]+dist[k][j]);  
 return dist;  
 }  
}  
class GFG{  
 public static void main(String[] args) throws IOException {  
 JFrame window = new JFrame();  
 window.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 window.setBounds(30,30,300,300);  
 window.getContentPane().add(new Levit());  
 window.setVisible(true);  
 Scanner scanner = new Scanner(new File("C:\\Users\\yaros\\SIAOD6\\src\\text.txt"));  
 int start = scanner.nextInt();  
 int end = scanner.nextInt();  
 int n = scanner.nextInt();  
 int[][] x = new int[n][n];  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 x[i][j] = scanner.nextInt();  
 }  
 }  
 int[][] inits = Levit.*schet*(x, n);  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 System.*out*.print(inits[i][j] + " ");  
 }  
 System.*out*.println();  
 }  
 System.*out*.println(inits[start][end]);  
 int answer = inits[start][end];  
 double time = System.*nanoTime*();  
 *paintgraf*(window.getGraphics(),n,x,start,end,answer);  
 System.*out*.println("Time: "+(System.*nanoTime*()-time)/1000000);  
 }  
 public static void paintgraf(Graphics g,int a,int[][] arr,int st,int en,int ans)  
 {  
 int[] x = new int[a];  
 int[] y = new int[a];  
 int angle = 360/a;  
 int i =0;  
 //g.drawOval(100,100,100,100);  
 while(i<a)  
 {  
 x[i] = (int)(140+50\*Math.*cos*(Math.*toRadians*(angle\*i)));  
 y[i] = (int)(140+50\*Math.*sin*(Math.*toRadians*(angle\*i)));  
 g.drawOval(x[i]-10,y[i]-10,20,20);  
 g.drawString(""+i,x[i]+10,y[i]-5);  
 i++;  
 }  
 for (int j=0;j<a;j++)  
 for (int j2=0;j2<a;j2++)  
 g.drawLine(x[j],y[j],x[j2],y[j2]);  
 g.setColor(Color.*GREEN*);  
 for (int k=0;k<a;k++) {  
 if (arr[st][en] == ans)  
 g.drawLine(x[st],y[st],x[en],y[en]);  
 else if (ans==arr[st][k]+arr[k][en]) {  
 g.drawLine(x[st],y[st],x[k],y[k]);  
 g.drawLine(x[k],y[k],x[en],y[en]);  
 }  
 }  
 }  
}

*Результат программы:*

*Входные данные задаются в виде файла, в котором первая строчка – начальная вершина, вторая строчка- конечная вершина, третья строчка – размерность матрицы и сама матрица смежности.*

1. *Входные данные:*

**

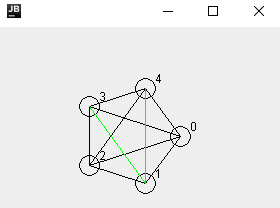
Результат программы:



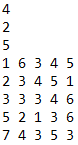
Также происходит вывод матрицы всех наикротчайших путей между всеми вершинами:



И вывод отрисовки самого графа с выделенным наикротчайшим путем между двумя вершинами:



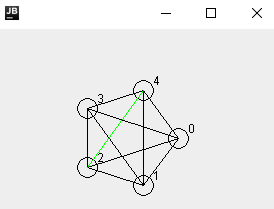
1. Входные данные №2(Матрица смежности остается такой же, меняются вершины таким образом, чтобы наикротчайший путь был прямой)



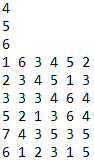
Результат программы:



Матрица всех наикротчайших путей остается такой же. Отрисовка графа:



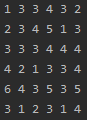
1. Входные данные №3(Количество вершин увеличено на 1):



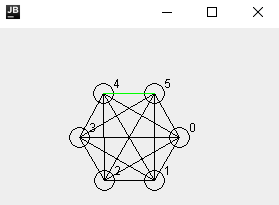
Результаты программы:



Матрица всех наикротчайших путей:



Выведенный граф:



*Таблица сравнения времени от числа узлов:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во узлов | 4 | 5 | 6 | 7 |
| Время работы программы | 31.023 | 32.17 | 43.3262 | 38.15 |

*Выводы:* реализовал алгоритм поиска кратчайшего расстояния между двумя вершинами ориентированного взвешенного графа. Предусмотрел задание графа в виде матрицы смежности\инциндентности, читаемой из файла. Разработал графический интерфейс пользователя с визуализацией графа и отображением кратчайшего расстояния между задаваемыми пользователем вершины. По результатам работы проанализировал временную сложность работы заданного алгоритма в зависимости от числа узлов и ребер графа. Время работы программы зависит от количества узлов и от выбранных узлов. Чем дольше путь от одной вершины к другой, тем дольше время выполнения программы. Программа реализована с отрисовкой графа, на которой отображен зеленым цветом наикротчайший путь. В работе представлена таблица сравнения времени выполнения программы от количества вершин(узлов). В конце получен корректный результат.